APPENDIX I

```
% Creating a network topology object

network_topo = topo('init');         % graphically place nodes on screen
addlink(network_topo);               % graphically connect up nodes
labelnames(network_topo);            % graphically label nodes

save network_topo;                   % save network_topo for future use


% Top level procedure to compute paths that optimize use of network capacity
% inputs:
%       D = traffic demand matrix
%         (retrieved from predictions stored in TMS Statistics Repository)
%       network_topo = topology object defining the network topology
%       P = network policy information
%          (matrix of reserved capacity, which indicates links whose use
%           is administratively prohibited or which should not be
%            completely allocated)
% outputs:
%       allocated_paths() = list of paths to set up, to TMS signalling system


C = capacity(network_topo);          % retrieve network topology information
C = C - P;

saved_C = [];
saved_SLA = [];
assigned_paths = [];
round = 0;

[SLA, S] = create_ordered_sla(D);

F = SLA(1)

for F = SLA',
        round = round +1;
        saved_C{round} = C;
        saved_SLA{round} = F;

        F % display the flow

        W = calc_weights('calcweight2',F,C);
        [dist, P] = floyd(W);
```

```
		path = findpath(P,F.i,F.j);

		assigned_paths{round}.path = path;
		assigned_paths{round}.flow = F;

		if (isempty(path))
			fprintf(1,'no path for flow:\n'); F
		else
			C = compute_residual_capacity('c - F.bw',path,F,C);
		end

	end

	function [W] = calc_weights(func,F,C)
	% function [W] = calc_weights(func,F,C)
	%
	% Compute the weights by calling func on each elt of C
	% func must be of the form double func(Flow F, Capacity_elt c, node i, node j)

	func = fcnchk(func);

	for i = 1:size(C,1)
		for j = 1:size(C,2)
			W(i,j) = feval(func,F,C(i,j),i,j);
		end
	end

	function [w] = calcweight2(F,c,i,j)
	% function [w] = calcweight2(F,c,i,j)
	%  basic weight calc

	if (0 == c)
		w = inf;
		return;
	end


	% rule out paths that can't hack it
	if (F.bw > c)
		w = inf;
		return;
	end
```

```
w = 1 / (c - F.bw) ;  % fill links with most capacity first

function [C] = compute_residual_capacity(func, path, F, C)
% function [C] = compute_residual_capacity(func, path, F, C)
%
% Update capacity characteristics in C to reflect flow F being
% allocated along path using function func
%    func should be of the form
%       C_element func(C_element c, Flow F)


if (length(path) <= 1)
        return;
end

func = fcnchk(func,'c','F');

index = 1;
src = path(index);
index = index + 1;

for index = index:length(path)
        dst = path(index);

        C(src,dst) = feval(func,C(src,dst),F);

        src = dst;
end

function [SLA, S] = create_ordered_sla(D)
% function [SLA] = create_ordered_sla(D)
% takes the demand matrix and returns a list of SLAs,
%      SLA of the form [ struct ; struct ; ... ] where struct is [BW, i, j]
%      S of the form [ [BW, i, j] ; [BW, i, j] ; ...]


S = [];

for i = 1:size(D,1)
        for j = 1:size(D,2)
                if (D(i,j) ~= 0)
                        S = [[D(i,j) i j ] ; S];
                end
        end
end
```

```
[Y, I] = sortrows(S,1);

S = Y(size(Y,1):-1:1,:);  % reverse order

SLA = struct('bw',num2cell(S(:,1)),'i',num2cell(S(:,2)),'j',num2cell(S(:,3)));

return;

function [path] = findpath(P,i,j)
% function [path] = findpath(P)
%
%

path = [];

if (i == j)
        path = [i];
        return;
end

if (0 == P(i,j))
        path = [];
else
        path = [findpath(P,i,P(i,j)) j];
end

function [D, P] = floyd(W)
% function [D, P] = floyd(W)
% given weights Wij, compute min dist Dij between node i to j
% on shortest path from i to j, j has immeadiate predecessor Pij

n = size(W,1);
if (n ~= size(W,2))
        error('Input W is not square??!!');
end

D = W;

P = repmat([1:n]',[1 n]);
P = P .* ~isinf(W);
P = P .* ~eye(n);

for k = 1:n
        for i = 1:n
```

```
                for j = 1:n
                        alt_path = D(i,k) + D(k,j);
                        if (D(i,j) > alt_path)
                                D(i,j) = alt_path;
                                P(i,j) = P(k,j);
                        end
                end
        end
k;
D;
P;
end
```